WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences

Your wildcard search against 2000 terms has yielded the results below

Search for additional matches among the next 2000 terms

starting with: SET$(SETEMP).P28-P88,P90-P90,P23-P27,P20-P22,P1-P18,P19-P19.

Search Results -

Terms	Documents
l1 and (multi near field$) and character near set$	3

Database:
US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Refine Search:

```
l1 and (multi near field$) and character
near set$
```

Clear

Search History

Today's Date: 10/21/2001

DB Name	Query	Hit Count	Set Name
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l1 and (multi near field$) and character near set$	3	L7
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l1 and (multi near field$)	30	L6
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	((text near string$) and word$ and (meaning or pronunciat$) and language$ and transliterat$).clm.	0	L5
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	((text near string$) and word$ and (meaning or pronunciat$) and language$ and transliterat$).ab.	0	L4
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	((text near string$ or word$) and (meaning or pronunciat$) and language$).ab.	750	L3
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	((text near string$ or word$) and (meaning or pronunciat$) and language$).ti.	42	L2
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	(text near string$ or word$) and (meaning or pronunciat$) and language$	9589	L1

WEST

Your wildcard search against 2000 terms has yielded the results below

Search for additional matches among the next 2000 terms

Generate Collection

Search Results - Record(s) 1 through 3 of 3 returned.

□ 1. Document ID: US 6115710 A

L7: Entry 1 of 3 File: USPT Sep 5, 2000

US-PAT-NO: 6115710
DOCUMENT-IDENTIFIER: US 6115710 A

TITLE: Portable and dynamic distributed transaction management method

DATE-ISSUED: September 5, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
White; John W.	Dallas	TX		

US-CL-CURRENT: 707/10; 707/101, 707/103R, 707/200, 707/3

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

□ 2. Document ID: US 5339392 A

L7: Entry 2 of 3 File: USPT Aug 16, 1994

US-PAT-NO: 5339392
DOCUMENT-IDENTIFIER: US 5339392 A

TITLE: Apparatus and method for creation of a user definable video displayed document showing changes in real time data

DATE-ISSUED: August 16, 1994

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Risberg; Jeffrey S.	Palo Alto	CA	94303	
Skeen; Marion D.	Palo Alto	CA	94306	

US-CL-CURRENT: 345/762; 345/765, 345/774, 707/501.1

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

□ 3. Document ID: US 4881197 A

L7: Entry 3 of 3 File: USPT Nov 14, 1989

US-PAT-NO: 4881197
DOCUMENT-IDENTIFIER: US 4881197 A

TITLE: Document composition system using named formats and named fonts

DATE-ISSUED: November 14, 1989

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Fischer; Addison	Naples	FL	33942	

US-CL-CURRENT: 707/530; 707/517, 707/522, 707/529, 707/542

Full | Title | Citation | Front | Review | Classification | Date | Reference

KWIC | Draw Desc | Image

Generate Collection

Terms	Documents
l1 and (multi near field$) and character near set$	3

Display 10 Documents, starting with Document: 3

Display Format: CIT Change Format

WEST

□ Generate Collection

L7: Entry 1 of 3 File: USPT Sep 5, 2000

DOCUMENT-IDENTIFIER: US 6115710 A
TITLE: Portable and dynamic distributed transaction management method

BSPR:
Application logic is most frequently written in "C" or COBOL program languages. User interface (e.g., screen format) control tables are defined and packaged separately from the application's load module, although there is strong coupling between the user interface definition and the application's data processing logic. Working storage contains data elements which are in a form directly usable by the application program.

DRPR:
FIGS. 40a-f are a flow chart depicting the edit language list (EL) procedure according to the preferred embodiment of the present invention;

DEPR:
The main storage logic 10 performs the computation and data manipulation for the application. Application program logic 32 pulls information from, and stores information on, data base(s) 18 in the system using a preselected subset of Structured Query Language (SQL) commands 20, although other languages could be employed. When the computation and manipulation are completed, the IET 26 begins its cleanup.

DEPR:
Application procedures contain no code related to either data import or data export. Application procedures are restricted to perform computation and data manipulation using data elements defined in working storage and optionally an interface to some information system. An information system supported by DAA is relational data base access via a well defined subset of Structured Query Language (SQL) commands. Other information systems could be employed by the application.

DEPR:
The isolation that the IET provides enables the application procedure to contain primarily application specific logic. The languages (primary and DBMS) used in application procedures are written in languages which are implemented consistently across all DAA platforms so as to ensure application portability.

DEPR:
The TDT consists of a variety of information such as application identification, version number, last transaction build date, help documentation file name, number of documentation languages supported, a pointer to first documentation language table entry, and an SQL support procedure entry point addresses for such functions as Connect, Commit, and Release. Map and procedure tables as well as menu and documentation language table entries also comprise part of the TDT.

DEPR:
Each of the map tables for application, menus, help, information and glossary panels contains such information as panel name, TMS screen map pointer, input/output procedure pointers, number of associated menu entries, and a pointer to the first associated menu entry. Each of the procedure tables contains such data as procedure name, procedure entry-point address, input/output/profile view table pointers, and SQL DB use flag. Each of the menu table entries contains such information as menu select code, panel entry pointer, panel procedure entry-point address, and displayable menu description string. Each of the documentation language table entries contains such specifics as documentation language name and documentation file name.

DEPR:
The TDF consists of a variety of types of records such as header, panel, menu, and procedure. The header typically contains a list of source, object, load and map libraries along with pertinent names, such as application and transaction-view names. Each panel is a record containing panel name and input/output processing procedure names. Each menu screen is a record containing menu name, menu panel name, input/output processing procedure names and displayable description string. Each procedure definition

is a record containing procedure name, input/output/profile view files names, and language type.

DEPR:
In most general case, the present invention provides the capability to develop interrelated applications and put these applications into service on multiple heterogeneous processors connected with heterogeneous communications facilities utilizing heterogeneous data bases. Currently the COBOL and C languages are being supported. It should be understood however, that other languages could be used in applications involving the present invention. Furthermore, while particularly DBMS is the primary data base management system employed in implementing the present invention, other data base management systems such as DS/1 could also be used. Lastly, it is contemplated that the present invention should not be limited to the TSO, Unix and OS/2 environment platforms currently supporting the present invention.

DEPR:
The distinguishing features of a preferred embodiment of the present invention include application portability, consistent user interface, dynamic application changes, and cooperative processing between heterogeneous computer architectures. Application portability means that an application written in one of the supported programming languages (e.g., COBOL or "C"), along with all related copy (or include) files, user interface panel definitions, documentation, transaction definition tables, views, etc., can be moved and installed, without source-level modification, on any supported target platform.

DEPR:
A transaction view is a collection of data items to be used by the collection of procedures, panels, and views that make up a transaction. These data items may be thought of COBOL Data Definitions or as C structures, although other languages may be used, that specify each of these data items. The transaction view is passed as working storage area to each procedure within the transaction for use in executing the logic of that procedure.

DEPR:
External views are implemented according to the ISO presentation protocol standard, which defines both a "source" language specification for data structures as well as the run-time data encoding algorithms.

DEPR:
REPORT.GENERATION. For a given combination of national language and transaction, there is a single "report.generation" which includes all application help information. Help functions may also invoke the general DAA help information, which is located on report=DAAHELP; generation=(language code).

DEPR:
Within the help dialog, every help panel (with the exception of Extended Help panels) provide function key access to Extended Help. The title of help panels contain the word "Help" and identify the panel or field to which it applies.

DEPR:
The TDF consists of the following types of records: header, first header extension, second header extension, panel, procedure, menu and language. Each TDF will have only one header, first header extension and second header extension, in that respective order. Each TDF will also contain at least one panel, procedure and menu record and they will be in no particular order. The presence of language records in a TDF is optional.

DEPR:
The OBJLIBN library/directory is used by GTD as an intermediate storage for object code versions of various elements of the particular transaction being defined. These intermediate objects include the TDT, the view member, and COBOL (or other language of choice) procedures as they are compiled. This library/directory is normally transparent to the user.

DEPR:
If the transactions are not operating on the proper system, the GTD runtime controller (IET) uses the system name on the panel and the DAA environment definition, also known as the Distributed Resource Controller (DRC), to determine the system type and routing specifications to route this transaction to the desired system. System name and type information (strictly design choice as to name of function, usually language dependent) is presented to the application via the following data items:

DEPR:
While in the HELP presentation, a user can request a definition of a word in text by placing the cursor on the word of interest and pressing the F2 key. F2 invokes the

Glossary function. If the word has been defined by the application designer in the GLOSSARY, the definition will be presented on the screen. After viewing the definition, the user can press the F3 key to return to the HELP documentation, and in turn press F1 to exit from HELP and return to the suspended application panel.

DEPR:
GLOSSARY SUPPORT FOR GTD APPLICATIONS. The Glossary function allows the application designer to define a set of words in the designated TIOLR report.generation so the user can interactively select the definition of a word they do not understand. Glossary words are specified in the TIOLR chapter "report.generation.GLOSSARY". Words within the Glossary can be one to eight characters long. Each word is specified as a section within the Glossary chapter. Each word description may be up to several pages long.

DEPR:
Once a word has been selected from the Glossary, function keys allow the user to scroll through the possible multiple pages of documentation for that word. To find other words in the Glossary the user can either use function keys to scroll to next or prior words within Glossary, or enter a particular word in the entry field. If a word is specified by either method, and is not found in the Glossary, the next word in alphabetical order will be shown.

DEPR:
Word definitions can be selected from the Glossary from the HELP panel, INFO panel, or GLOS panel by placing the cursor on the word of interest and pressing F2. Once the user has completed a review of the Glossary, he can press F3 to return to the prior panel.

DEPR:
A COBOL procedure within a GTD application is a standard structured COBOL member. The application designer can use any of the basic COBOL functions, and reference other programs and procedures, using standard language facilities of the environment where the transaction is intended to run. The COBOL procedure is intended to be a COBOL II structured COBOL member with minimal references to other facilities so the application can be portable to any of several environments.

DEPR:
GTD APPLICATION PROCEDURE STRUCTURE. The GTD system primarily supports COBOL and C applications. The GTD system will support Assembly Language Code (ALC) implementation, but these implementations require manual correlation of information not required in COBOL and C environments. For example, in the ALC environment, a DSECT must be maintained that is byte level compatible with the COBOL or C transaction view generated for view and panel references.

DEPR:
GTD Documentation Report Definition--Language of Preference. GTD applications documentation is stored on TIOLR for the transaction user to access in one of the following ways:

DEPR:
The documentation can be made available in multiple languages. The language of the documentation displayed depends on what the user has selected as his first and second language of preference (LP). The selections are made using the ITSS "Language of Preference" transaction, TSSLP.

DEPR:
The selection criteria of the report.generation (based upon language) is:

DEPR:
The work area variables are initialized with the parameter file variables, and the program proceeds to Block 5710, where it loads system-dependent environment information that applies to all users on the platform as well as information that is configurable by the user. The environment information that applies to all users on the platform includes: C compiler option, C compiler name, DAA system object/load library pathnames, DBMS software object/load library pathnames, DBMS copy source pathnames, DBMS pre-compiler pathname. This information is obtained from environment variables defined for all users. Other system-dependent environment information that is hard-coded into the program includes the specific names of all utilities, the specific options required by those utilities, the names of system, DAA and DBMS libraries. The system-dependent utilities include: C compiler, DBMS pre-compiler, DBMS to DBMS language translator, COBOL compiler, Linkage editor, editor, show file, copy, move, kill task utility, panel compiler, panel screen formatter (interpreter), panel editor, COBOL data map generator (compiler), COBOL to C translator.

DEPR:
It should be remembered that while the preferred embodiment depicts COBOL and C, the

scope of the invention should not be limited in any way to these two languages. Additionally, an alternative to using the environment variables is to have a system environment information file which would have the above information, and possibly other information that is currently hard-coded in the program, stored in the file. This file can be loaded by GTD from a predefined location each time it is invoked, in much the same way as it loads environment variables in the preferred embodiment of the present invention.

DEPR:
If GTDSEL does not equal 1 at Block 6390, the procedure checks to see if GTDSEL equals 2, meaning the user requested the generate procedure or transaction view menu option. When GTDSEL equals 2, the generate procedure or transaction view procedure (GC, FIG. 44, beginning at Block 6900) is called. When the GC procedure is complete, the program returns to Block 6260, to see if the user requires any other procedures from this panel.

DEPR:
If GTDSEL does not equal 2 at Block 6430 the program checks to see if GTDSEL equals 3, meaning the user requested the generate panel menu option. When GTDSEL equals 3, the generate panel procedure (GM FIG. 47, beginning at Block 9030) is called. When the GM procedure is complete, the program returns to Block 6260, to see if the user requires any other procedures from this panel.

DEPR:
If there is no edit menu hierarchy request, the procedure checks to see if there is an edit language list request, GTDSEL=5 (Block 102710). If there is, the edit language list procedure (FIG. 40, Block 113070) is performed and the procedure continues at Block 102810. If there is no edit language list request, the procedure checks to see if there is an edit transaction view request, GTDSEL=6 (Block 102750). If there is, the fully qualified name of the transaction view file is built and the edit procedure (FIG. 33, Block 43860) is called to bid an editor. Once the edit is complete, or if there was no edit transaction view request, the procedure proceeds to Block 102810.

DEPR:
GTD GET TDF; GTDT PROCEDURE. According to FIGS. 24a-r which depict the preferred embodiment of this procedure, the GTDT procedure loads a TDF file into memory. The header records are stored in appropriate areas in the transaction view and the panel, procedure, language, and menu entry component records are stored in the STDE table. If no TDF file exists, then the memory areas are initialized with default TDF file values. The format of the TDF file records, and consequently the format of the transaction view header record work areas, as well as the STDE table entries, are documented in the TDF documentation.

DEPR:
The procedure entry fields are validated (Block 107450). The validation includes translating the procedure name, input, output and profile views, and procedure type, data base procedure flag, and debug flag into upper case. The profile view name, data base procedure flag, and debug flag are blanked out when a procedure is defined to have a type of EXTERN. The data base flag and debug flag are blanked out when a procedure is defined to have a type of ENTRY. All fields, data base procedure flag and debug flag, which have a required value of either "Y" or "N", are redefined to have a value of "N" if the value defined by the user is not "Y". The procedure language type is defined to be C if the language defined by the user is not COB2, ALC, or C.

DEPR:
GTD EDIT: ECE PROCEDURE. Looking now at FIGS. 34a-b, which depict a flow chart of the preferred embodiment of this procedure, the GTD edit procedure begins by searching for the requested edit procedure name in the STDE table. Block 109530 checks to see if the search was successful. If it was not, the program exits the procedure without error. If the search was successful, the procedure checks to see if the procedure language is COBOL. If it is not a COBOL program, the procedure proceeds to Block 109750. If it is a COBOL program, the procedure builds a COBOL procedure file name (Block 109580), and verifies the procedure file exists (Block 109670). If the file exists (Block 109675), the procedure jumps to Block 109750. If not, the procedure saves the procedure name in gtdpmem (Block 109680), and generates (Block 109690) a skeleton program using the GCOBPROG procedure (FIG. 35, Block 7400). The program then jumps to Block 109900, to do the edit.

DEPR:
Block 109750 checks to see if the procedure is in C language. If it is not, the procedure continues at Block 109930. If it is in C, the procedure builds a procedure file name (Block 109760), and verifies the procedure file exists (Block 109850). If the file exists (Block 109855), the procedure proceeds to Block 109860. If not, the procedure saves the procedure name in gtdpmem (Block 109860), and at Block 109870

generates a skeleton progr using the GCPROG procedure (FIG 5, Block 114950).

DEPR:
EDIT LANGUAGE LIST. This procedure is another third level procedure which provides a display panel which includes information from the TDF language record type. This information characterizes a language entry definition and is displayed on the panel in a list. Since more language entries may be defined than may be displayed at one time, facilities are provided to allow navigation through the language list. Functions are also provided for adding new language definitions, modifying current language definitions, deleting current language definitions, and displaying current definitions.

DEPR:
When the user requests HELP documentation, GLOSsary support, or INFOrmation, the Language of Preference field determines the appropriate documentation to present to the user. If the documentation does not exist in the user's language of preference, English documentation will be presented, if available.

DEPR:
The GTD system supports applications that use HELP, GLOS, and INFO documentation to describe the functionality of the application. This documentation is maintained as part of the TDT. As the TDT is initially built, the designer can specify the report and generation to designate the documentation location. This feature can be extended to support multiple languages through the Edit Language List facility. The Edit Language list facility allows the designer to edit a list of languages and the associated report.generation for the documentation. The application designer can then provide documentation in the user's language of preference.

DEPR:
GTD 1.5 presents the designer with a list of languages specified for this TDT. Each language entry includes a report and generation specification to identify where the documentation resides for that language. The designer can add, change, or delete entries within this language list, by entering `A`, `C`, or `D`, respectively, and modifying the entries as necessary.

DEPR:
The GTD edit language list procedure begins by scanning the select field for the locate command input by the user. The locate name is saved in the dname field of the transaction view. This locate name is used as the initial language entry to be displayed in the edit list. If there is no locate command, dname is set to blank (Block 113100). The following display area variables are blanked (Block 113110): select, variable gtdsela; name, variable dnamea; report, variable dipnm; and generation, variable dopnm. Index, j, and pointer E are initialized to the beginning of the STDE table (Block 113180).

DEPR:
Blocks 113185 through 113260 constitute a process by which the initial language list item to be displayed on the screen is located in the STDE table. Then the initial language name, and as many subsequent languages names as will fit on the display, are placed in language list display fields with the corresponding language characterization information from the STDE table for each language entry displayed. The process is finished when all language names up to the end of the list, or all languages names which will fit on the screen, and their characterization data have been placed in the display fields.

DEPR:
Decision Block 113185 checks to see if e is pointing past the end of the STDE table. If it is, the procedure jumps to Block 113270. If not, the procedure checks to see if j is greater than or equal to the number of language entries(16) which will fit on a screen (Block 113200). If j is greater than or equal to 16, the procedure jumps to Block 113270. If not, the procedure checks to see if the current language (STDETYP=L) is an STDE table language name, STDENAME, greater than or equal to the next language name, dname, requested by the user (Block 113205). If the current language name is not greater or equal to the request language name, the procedure jumps to Block 113260. If the current language name is greater or equal to the next language name, the jth item in the edit list is initialized to the following values: ;name, variable dnamea=STDENAME; report, variable dipnm=STDEIPNM, and generation, variable dopnm=STDEOPNM, (Block 113210). The procedure increments index j (Block 113240), increments pointer e to point to the next STDE entry (Block 113260), and then returns to Block 113185 to continue processing.

DEPR:
Blocks 113285 through 113475 constitute a process by which each user request for each language list item is examined. If an add, delete, or change, request is identified then the appropriate task is performed. The process is complete after the requests have been

identified and performed f each language entry in the list

DEPR:
J is initialized to zero (Block 113280) and the procedure checks to see if j is greater than or equal to the number of language entries (16) which will fit on a screen (Block 113285). If j is greater or equal to 16, the procedure proceeds to Block 113480. If not, the procedure checks to see if there is an add language request, gtdsela=`a` or `A` (Block 113310).

DEPR:
If there is an add request, a language is displayed requesting the language characterization information as described in the language record type of the TDF. The input values are validated. This typically involves translating the language, documentation report, and documentation generation names to upper case. After the input values are validated an attempt is made to add the new language entry to the STDE table for later storage in the TDF. The attempt will fail if the language name provided is blank, if the STDE table is full, or if the language name provided already exists in the table. If no error is detected, the language entry is inserted in the STDE table ascending language sequence. The STDE entry count, nent, is incremented. Once the language is added, the procedure jumps to Block 113470. If there is no add request, the procedure checks to see if a change language request gtdsela=`c` or `C` (Block 113360).

DEPR:
If there is a change request, the change request language name is verified. A change request can not be performed if the language is not defined in the STDE table. If the language is listed in the STDE table, the values in the STDE language entry are then copied into the workarea fields which correspond to STDE entry and a language is displayed requesting changes for the language characterization information. The input values are validated as described for an add language request (Block 113310) and moved back into the STDE entry to complete the change. The only change the user is not permitted to perform with this implementation is a change to the language name. Once the language is changed the procedure proceeds to Block 113470. IF there is no change request, the procedure checks to see if there is an delete language request, gtdsela=`d` or `D` (Block 113410).

DEPR:
If there is a delete request, the delete request language name is verified. A delete request can not be performed if the language is not defined in the STDE table. If the language is listed in the STDE table, then the language is deleted by overwriting the deleted entry with the later entries in the STDE table. The STDE entry count, nent, is decremented. Once the language is deleted the procedure moves to Block 113470. If there is no delete request, the procedure increments j (Block 113470) and returns to Block 113285.

DEPR:
In summary, the following table types are constructed by GTD through the CT procedure: TDT anchor block (consists mainly of pointers to other TDT tables); transaction panel entries table, TPETAB, with one TPE per panel; transaction code entries table, TCETAB, with one TCE per referenced executable procedure entry point; transaction menu entries table, TMETAB, with one TME per menu panel; transaction language entries table, TLETAB, with one TLE per language supported; and the data base sync table referencing SQL data base support routine entry points.

DEPR:
Building this TDT object module can be accomplished different ways. On UNIX, the CT procedure, as flow charted here, builds a C language source file that is later compiled by the C compiler to produce a link-editable object module. On MVS a CT version exists that constructs 370 linkable objects modules directly, without the aid of a C compiler.

DEPR:
The CT procedure involves building and compiling the TDT. The TDT is built in the following manner. First the STDE table is inspected to ensure that all the definitions are complete and consistent by determining if any panels or procedures have been left undefined (i.e., without a source file). Then all panels, menus, procedures, languages, and views are counted and a table built containing the names for each type of component. Once all the components are accounted for, a source procedure is built containing source language statements which define the customized TDT information. In the case of the current implementation for UNIX, this source file is built as a C source file. The TDT source generation proceeds by generating a standard source prefix, external name definitions for the panels, views, procedures; generating the TDT, TPETAB, TCETAB, TMETAB, TLETAB, and DB synchronization function address table; and then generating a standard source suffix. The standard source prefix contains files, function prototypes, and external definitions common to all TDT source. The standard source suffix contains a small procedure, DLITCBL, that is called by the transaction sub-system main module upon

execution of the transacti DLITCBL, in turn, calls IET, p ing the address of the TDT and any input/output control block passed by the transaction sub-system main module. Once the TDT source file is built, it is compiled using a standard compiler. In the case of an UNIX implementation, a C compiler is used.

DEPR:
This section begins by zeroing out each counter: NP for panels, NC for procedures, NL for languages, NV for views, and NM menu components or entries in STDE table. The DBMS procedure found flag is initialized to zero (Block 18630), the panel table, TP is blanked (Block 18660), and the pointer, e, is set to the first STDE table component name (Block 18700).

DEPR:
Decision Block 18960 checks to see if the component type, STDETYP, of the component indicated by e is equal to an `L` (indicating a language entry definition). If so, the language counter, NL, is incremented. Decision Block 18970 checks then to see if the component type, STDETYP, of the component indicated by e is equal to an "E" (indicating a menu entry definition). If so, the menu counter, NE, is incremented. At Block 18980, the procedure increments pointer e to the next STDE table component entry and then jumps to Block 18705.

DEPR:
Block 19310 initializes buffer TDTBUF to blanks, and builds a C language external entry in the buffer as an "extern char", panel name from table at index loc i, "[ ];" (Block 19320). The line in the buffer is terminated by placing a new line character after the data (Block 19370). In Block 19380, the write buffer TDT source file WTS procedure (FIG. 53, Block 25500). The procedure then increments the index, i, in Block 19390, and proceeds to Block 19135.

DEPR:
In Block 19460, index, i, is initialized to zero. Decision Block 19465 checks to see if index, i, is equal to the number of views. If so, the procedure jumps to Block 19620. Otherwise, the buffer, TDTBUF, is set to blanks (Block 19470), and a C language external view entry is built in the buffer as "extern struct VMOH", view name from table, TV, at index loc i, "[ ];" (Block 19480).

DEPR:
Blocks 19620 through 19730 generate EXTERN entries for procedures. The pointer, e, is set to the first STDE table component name (Block 19620), and then the procedure checks to see if e points past the end of the STDE table component name table. If so, the procedure jumps to Block 19800. If not, the procedure next checks to see if the component indicated by e is a language (STDETYP=C) entry type, with an external procedure type (STDECTYP=EXTERN). If not, the procedure continues at Block 19730. If it is, the procedure initializes the buffer, TDTBIF, to blanks (Block 19640).

DEPR:
The C language external procedure entry is built in the buffer as "extern int", with the procedure name from the STDE table (STDENAME) at index location i, "()" (Block 19650). The line in the buffer is terminated by placing a new line character after the data (Block 19700) and the procedure performs (Block 19710) the write buffer TDT source file WTS procedure (FIG. 53, Block 25500). (Block 19730 increments e to the next STDE table component entry, and the procedure then returns to Block 19625.

DEPR:
This section starts at Block 19800 by initializing the large buffer to blanks. The buffer C language TDT table structure definition header is built in the buffer as "struct TDT IETTDT=(0 (Block 19810). The C source to initialize the TDTID is built in the buffer using the value of gtdmmem (Block 19830). The C source to initialize the TDTRELN is built in the buffer using the value of gtdreln (Block 19870). The C source to initialize the TDTGTIME is built in the buffer using the current clock time recorded as seconds since 07/01/01 (Block 19960).

DEPR:
The C source to initialize the TDTNTLE is built in the buffer using the value of NL as the number of language entries (number of TLEs) (Block 20610).

DEPR:
Block 20660 checks to see if the number of language entries is greater than zero (Block 20660). If it is, the C source to initialize the TDTATLE is built in the buffer and stored as the pointer to the first entry in TLETAB. Otherwise, the C source to initialize the TDTATLE is built in the buffer and stored as zero, or a NULL pointer reference. indicating that there is no language table, TLETAB.

DEPR:

The C language TDT table structure definition, IETTDT structure termination data is built in Block 20720. The procedure then performs the write large buffer TDT source file WTS procedure (FIG. 53, Block 25500) to complete this section.

DEPR:
Next Block 20860 prepares to build all the panel entries by initializing index, i, to zero. Block 20865 checks for the end of the panel definitions by determining whether i is equal to NP. If it is, control is passed to Block 21920. Otherwise, the large buffer is initialized to blanks (Block 20870), and the C language TDT structure array element prefix is built (Block 20880).

DEPR:
The C source to initialize TPENTME (Block 21740) and TPEATME (Block 21790) to zeros are built in the buffer. The procedure builds the C language TDT structure array element suffix (Block 21840) and performs (Block 21890) the write buffer TDT source file WTS procedure (FIG. 53, Block 25500). Index i is incremented (Block 21900) and the procedure jumps to Block 20865.

DEPR:
The procedure initializes the large buffer to blanks (Block 21920), and then builds the C language TPETAB table structure array definition suffix in the buffer (Block 21930). The procedure then performs (Block 21940) the write buffer TDT source file WTS procedure (FIG. 53, Block 25500).

DEPR:
The C language TCETAB table structure array is built in the buffer using the value of NC as the number of procedures (TCE) structure elements in the TCETAB table (Block 22020).

DEPR:
The procedure initializes the large buffer to blanks (Block 22100), and then builds the C language TDT structure array element prefix (Block 22110).

DEPR:
Block 22520 checks if the procedure component is a DBMS type procedure, STDECDB2=Y. If it is a DBMS type, the procedure builds the C source in the buffer to initialize TCEDBTYP to 1, flagging it as a database procedure. If it is not, the procedure builds the C source to initialize TCEDBTYP to zeros, flagging it as a non-database procedure. The procedure builds the C language TDT structure array element suffix (Block 22610) and performs (Block 22650) the write buffer TDT source file WTS procedure (FIG. 53, Block 25500). Pointer e is incremented (Block 22670), after which the procedure returns to Block 22085.

DEPR:
At this point, the CT procedure creates source to build the menu table, TMETAB. Here too, there will be one entry per menu. The procedure initializes the large buffer to blanks (Block 22700) and builds a C language, TMETAB, table structure array definition suffix in the buffer (Block 22710). At Block 22710 the procedure then performs the write buffer TDT source file WTS procedure (FIG. 53, Block 25500).

DEPR:
Blocks 22790 through 23420 generate the TMETAB. This section begins at Block 22790 by initializing the large buffer to blanks. The C language TMETAB table structure array is then built in the buffer using the value of NE as the number of procedure (TME) structure elements in the TMETAB table (Block 22800). The procedure performs (Block 22830) the WTS procedure (FIG. 53, Block 25500) and then initializes the pointer, e, to the first STDE table component table. (Block 20865 checks to see if e points past the end of STDE table component name. If it is, the procedure jumps to Block 23400. Otherwise, the procedure (Block 22870) checks to see if the component type of the component indicated by e is a menu entry definition (STDETYP=E).

DEPR:
Block 22890 builds a C language TDT structure array element prefix. The C source to initialize TMESEL is built using the value of STDENAME in the STDE table name table entry indicated by index, e (Block 22920). (Block 22180 locates the panel name in the panel table with the same name as the menu output panel indicated by pointer e. Index j is initialized with the number of panels encountered in the panel table before the desired panel is located.

DEPR:
Block 23300 builds the C language TDT structure array element suffix and performs (Block 23350) the WTS procedure (FIG. 53, Block 25500). Pointer e is incremented (Block 23380) after which the procedure returns to Block 22085.

DEPR:

The large buffer is initialized to blanks (Block 23400), a C language TMETAB table structure array definition suffix is built in the buffer (Block 23410), and the procedure performs (Block 23420) the WTS procedure (FIG. 53, Block 25500).

DEPR:
The next portion of the CT procedure Blocks 23490 through 23870) generates the transaction language table, TLETAB. As noted by the flow chart, no transaction language entries will be built if none are defined. (Block 23490 checks to see if the number of language entries, NL, are greater than zero. If they are not, the procedure advances to Block 23950. If they are, the procedure initializes the large buffer to blanks (Block 23550).

DEPR:
Block 23510 builds the C language TMETAB structure array definition using ne as the number of procedure (TME) structure elements in the TMETAB table. The procedure performs the WTS procedure (FIG. 53, Block 25500) and pointer e is incremented to the first STDE table component name (Block 23380).

DEPR:
Block 23575 checks to see if e points past the end of the STDE table component table. If it does, the procedure jumps to Block 23580. If it does not, the procedure checks to see if the component type (STDETYP) of the component indicated by e is a language entry definition. If not, the procedure jumps to Block 23830. If it is, the procedure initializes large buffer to blanks (Block 23590) and builds a C language TDT structure array element prefix (Block 23600).

DEPR:
Block 23630 builds the C source in the buffer to initialize TLENAME using the language name in the STDE table name table entry indicated by index, e.

DEPR:
The procedure initializes the large buffer to blanks (Block 23850), builds the C language TLETAB table structure array definition suffix (Block 23860), and performs (Block 23870) the WTS procedure (FIG. 53, Block 25500).

DEPR:
This section of the procedure begins at Block 23960 by initializing the large buffer to blanks and builds the C language DB synchronization function address table structure definition (Block 23960).

DEPR:
The procedure builds the C language DB synchronization structure initialization suffix (Block 24210), and performs (Block 24230) the WTS procedure (FIG. 53, Block 25500).

DEPR:
During the build view process the C transaction view file (.h) is regenerated from the COBOL transaction view file to maintain consistent transaction views between COBOL and C language procedures.

DEPR:
Blocks 27020 through 27160 set up an internal table of transaction view field information. This process is essentially a compiler for the transaction view COBOL source which encodes the field displacement, type and length information output from the compile, and places that information into an internal table. The internal table contains for each field: the field displacement relative to the beginning of the transaction view, the field type, the field length, and the field occurrence multiple. This information is used when creating the VMOF table entries and as input to the COBOL/C translation utility which translates the COBOL transaction view into a C version of the same transaction view. Another translation utility may be employed which does not use the internal table as input. The method of the preferred embodiment used here, is efficient because it does not require any duplication of effort between generating the field offset, length, etc. information and generating the C version of the transaction view. The two views are required to result in the same memory usage and allocation for each language version of the transaction view.

DEPR:
If the number of fields is greater than zero, Block 29430 initializes a large buffer to blanks. Block 29440 builds in that buffer C source language for a VMOF table structure array definition using the number of fields as the number of VMOF structure elements in the table. The values of the view table index number is used to generate VMOF table names as VMOFnnnn where nnnn contains the index value. Block 29490 writes the buffer TDT source file using the WTS procedure. Block 29510 initializes an index to point to the first name in the field table.

DEPR:
Moving now to FIGS. 57a-d, which depict the preferred embodiment of the CC procedure, the compile procedure initiates a compile process for each procedure in the application. It examines each procedure in the STDE table and calls the appropriate compiler for the procedure. In the current implementation only C and COBOL compilers are supported, however, any language may be used which can produce linkable object code.

DEPR:
The CC procedure begins by initializing E to point to the beginning of the STDE table (Block 33380). Decision Block 33420 checks to see if E points past the end of STDE table. If it does, the program jumps to Block 33600 and sets the message field, GTDMSG to "PROCEDURE COMPILE COMPLETE" at which time the program exits the CC procedure (Block 33610). If E does not point past the end of STDE table, the STDE entry is checked to see if it is a procedure entry, STDETYP=C (Block 33425). If the STDE table entry is not a procedure, execution continues at Block 33520. If the STDE table entry is a procedure, the procedure language is checked to see if the procedure is written in the C language, STDECTYP=C, (Block 33426). If the procedure language of the STDE entry is not the C language then the program jumps to Block 33520. If the procedure language of the STDE entry is the C language, Block 33430 calls the compile C procedure: CCO (FIG. 58, Block 33690).

DEPR:
Decision Block 33520 determines if the STDE procedure entry is a COBOL procedure STDECTYP=COBOL (Block 33520). If the procedure language is COBOL, at Block 33530 the program calls compile C procedure, CCOBO (FIG. 59, Block 36270). If not, the program jumps to Block 33580. Decision Block 33535 determines if the compile is successful. If the compile is successful, the program jumps to Block 33580. If the compile is not successful, error message field GTDMSG is set to "PROCEDURE COMPILE FAILS" (Block 33540) and the program returns from the CCOBO procedure with an error (Block 33550).

DEPR:
The BT procedure begins by rewinding the GTD output file (Block 40260), sets up the link editor parameter values and application input file with link editor files to be included in the link editor procedure (Block 40580) identifies the DBMS procedure types, if any, and procedure language types (Block 41790), sets up names of language, DBMS (if any), DAA system, operating system libraries and object files to be included in the link (Block 42060).

DEPR:
The Transaction Definition Table, or TDT, as used throughout this description, consists of various tables of information stacked together so that the IET can easily access the appropriate information at run-time for the transaction procedures. As seen in FIG. 4, the TDT consists of the following table types, described in further detail below: TDT overhead 36, list of panel map entries (TPE) 44, list of procedure entries (TCE) 46, list of menu entries (TME) 48, list of language entries (TLE) 50 defining the various languages (i.e., English, French, German, etc.) in which the documentation is available, and list of SQL data base processing procedures (DBSYNC) 52.

DEPR:
TRANSACTION LANGUAGE ENTRIES (TLE). The TLE list 50 is an array of TLEs each describing a language (i.e., English, French, German, etc.) in which the documentation is available, and containing the following fields:

DEPR:
TLENAME is the eight character language name.

DEPR:
The services the IET provides for the application which include menu handling, help and documentation handling, application profile management, cooperative processing and message/screen formatting isolate the application from the platform software by allowing the IET to handle any differences in platform software and hardware related to interface differences for file input/output, screen formatting and message handling. The IET implementation for each platform provide isolation for the application program. Further isolation is maintained through the use of common high level languages and a common embedded DBMS language interface for data base applications on all platforms. It is a result of the isolation from the hardware and software-dependent interfaces and the use of common language implementations which allow application portability across multiple execution environments.

DEPR:
If, however, the test yields a positive decision, the IET moves to Decision Block 150, where it checks if the sending IET's character set is different from the receiving IET'S character set (e.g., ASCII versus EBCDIC), by comparing the character type field of the IETI header with a the receiver's type. If the IET header is from an EBCDIC system, and

the local IET is ASCII, the IET interface converts all IETI character fields to ASCII (Block 160).

DEPR:
At Block 470, the IET checks the local machine character set for type of ASCII. If it is not an ASCII machine, the IET interface skips the next step and moves to Block 510. If it is an ASCII machine, the IET interface converts the data stream to ASCII code (500), and then moves to Block 510.

DEPL:
LANGUAGE: (one instance per language entry)

DEPL:
As other languages are employed, additional types of procedures may be identified.

DEPU:
STDELOPT, 60, Language options.

DEPU:
STDENAME, 8, Language name.

DEPV:
an application can be implemented on a variety of user environments, including variances in national language, human interface hardware, and user preference.

DETL:
Where: rpt = report name defined by LP gen = generation name defined by LP GLOSSARY = constant chapter name word = word being defined page = arbitrary page name(s)

DETL:
COB2 COBOL procedure. Will be processed by the COBOL II compiler using GTD options 4.5, 4.6, 4.8, and 4.9. C C procedure. Will be processed by the C compiler using GTD options 4.5, 4.6, 4.8, and 4.9. ALC Assembly language procedure. EXTERN External procedure. Procedure is candidate to be LINKed-to by a procedure within this transaction ENTRY Procedure contained within another procedure in the transaction.

DETL:
iefl.sub.-- helpblk iefl.sub.-- proc 8 iefl.sub.-- reln 4 iefl.sub.-- lookahd 64 iefl.sub.-- gdate 8 iefl.sub.-- gtime 8 iefl.sub.-- termdef 16 iefl.sub.-- ainput 4 iefl-tvwend 8 NAME LENGTH DEFINITION gtdhmenu 8 main menu name gtdhhelp 8 help panel name gtdhinfo 8 inf o panel nane gtdhglos 8 glos panel name gtdhimnu 8 information menu name gtdhpvw 8 profile view name gtdhmap 8 gtdtitl 32 gtdpmem 8 field containing name of an application procedure or menu gtdmsg 32 output field used to contain informational or error messages for benefit of the user zcmd 80 zluser 8 THESE FIELDS CORRESPOND TO THE DATA PORTION OF THE TDF HEADER NAME LENGTH DEFINITION gtdhrtyp 1 TDF GTDHRTYP record type gtdhmem 8 gtdhmtyp 1 TDF GTDHMTYP TDT type gtdhdr 8 documentation report name gtdhdg 8 documentation generation name gtdappl 4 appiication name gtdreln 4 tdt release number gtdllib 46 load library gtdlmem 8 load module member name gtdolib 46 object library gtdomem 8 object module member name gtdslib 46 field used to contain the pathname for all the application source procedures (also transaction view data sets) gtdcmem 8 name of transaction view gtddlib 46 data base library name gtddinem 8 data base library member name gtdnlib 46 control library path gtdnmem 8 control library member gtdleopt 50 linkage editor options gtdgof 1 generation option flag gtdglf 1 generation lookahead flag gtdrsvd2 THESE FIELDS CORRESPOND TO THE DATA PORTION OF THE TDF FIRST EXTENSION HEADER NAME LENGTH DEFINITION gtdhexl 2 record type gtdhexln 2 extension type gtdcdsnl 46 extra copy/header library path gtdcdsn2 46 extra copy/header library path gtdcdsn3 46 extra copy/header library path gtdcdsn4 46 extra copy/header library path gtdxlib 46 debugger symbols gtdexlf THESE FIELDS CORRESPOND TO THE DATA PORTION OF THE TDF SECOND EXTENSION HEADER NAME TYPE LENGTH DEFINITION NAME LENGTH DEFINITION gtdhex2 2 TDF record type gtdhex2n 2 extension type gtdldsnl 46 extra load library pathname gtdldsn2 46 extra load library pathname gtdldsn3 46 extra load library pathname gtdldsn4 46 extra load library pathname gtdldsn5 46 extra load library pathname gtdldsn6 46 extra load library pathname gtdex2f INPUT/DISPLAY FIELDS NAME LENGTH DEFINITION gtdsel 40 field used as user input usually to obtain numbered menu item selections gtdsel0 8 gtdsell 8 gtdsel2 8 gtdsel3 8 gtdmenul 8 gtdmenu2 8 gtdmenu3 8 dname 8 first name to display tdtbuf 400 zuser 8 user id zdate 8 formatted data of program entry ztime 5 formatted time of program entry gtdseli 1 gtdselo 1 gtdselp 1 gtdsela 16 input area select code for edit list: panel, procedurel menu display lines dnamea 8 A 16 display areas for component names: panels, procedures, menu, language dipnm 8 0 16 display areas for input procedure, input view, dopnm 8 0 16 display area for output procedure, output view, dpvnm 8 0 16 display area for profile view dm0ds 16 0 16 display area for level 1 menu sel/descriptions one for each menu entry in display

list dm1ds 16 0 16 display area for level 2 menu sel/descriptions one for each menu entry in display list dm2ds 16 0 16 display area for level 3 menu select/descriptions one for each menu entry in display list dm3ds 16 0 16 display area for level 4 menu select/descriptions one for each menu entry in display list gtdcty 8 0 16 one for each procedure entry in display list, Y = data base gtddb2 1 0 16 display area for procedure data base flag, Y = data base application, one for each procedure entry in display list gtdxpd 1 0 16 display area for procedure debug flag, Y = debug compile options are to be used, one for each procedure entry in display list gtdmmem 8 name of TDF file gtdmlib 46 pathname for maps and TDF files should be maximuin length for most restrictive platform COPY OF ONE INSTANCE OF A TDF COMPONENT AND STDE TABLE ENTRY (generic for all types) NAME LENGTH DEFINITION gtdetyp 1 TDF component record type gtdename 8 panel, procedure, menu, or language name gtdesel 8 menu selection value gtdeipnm 8 input view for procedures input procedure for panels panel to invoke for menus report name for languages gtdeopnm 8 output view for procedures output procedure for panels procedure to invoke for menus generation name for languages gtdepvnin 8 procedure view name gtdedesc 32 panel, procedure, menu description gtdersvl 4 gtdeccbm 1 gtdecdb2 1 procedure data base flag gtdecxpd 1 debugger flag gtdectyp 8 procedure type gtdelopt 60 compiler language options gtdersv2 OTHER INPUT/DISPLAY FIELDS NAME LENGTH DEFINITION gtdvmem 8 gtdmvwl gtdcvwl gtdaname 32 statitem 20 statstat 60 srcsid 8 source system id srclid 48 source pathname srctdt 8 source tdt name srcuid 8 source user id srcupw 8 tgtsid 8 target system id tgtlid 48 target pathname tgtuid 8 target user id tgtupw 8 tgtsw 1 target rsvrdl1 7 rsvrdl0 STDE TABLE entry NAME LENGTH DEFINITION gtdetyp 1 TDF component record type gtdename 8 panel, procedure, menu, or language name gtdesel 8 menu selection value gtdeipnm 8 input view for procedures, input procedure for panels panel to invoke for menus report name for languages gtdeopnm 8 output view for procedures output procedure for panels procedure to invoke for menus generation name for languages gtdepvnin 8 procedure view name gtdedesc 32 panel, procedure, menu description gtdersvl 4 gtdeccbin 1 gtdecdb2 1 procedure data base flag gtdecxpd 1 debugger flag gtdectyp 8 procedure type gtdelopt 60 compiler language options gtdersv2

DETL:

IEF1-SAVE wsave 72 SAVE area available for use by ALC GTD applications. EF1-APPL wappl 4 Application (APPL) ID of active transaction. Used by profile manager to identify activity for this transaction in profile space. APPL ID is also presented on "Show Active Application Display" under TRAN=A. IEF1-RSVRDO wrsvrdo 4 Reserved. IEF1-TRAN wtran 8 Transaction code of active transaction IEF1-PANEL wpanel 8 Panel name of active panel. IEF1-TERM wterm 8 Terminal name (under IMS, LTERM name. Identifies terminal from which input has been received. It received from TDT will be blank. IEF1-USER wuser 8 User ID. Set at transaction initialization to identify user of this particular transaction and application. IEF1-PSWD wpswd 8 Password. Is available to application to verify user is authorized to perform various functions of this application. This is not DAA network log-on password. IEF1-SYSTEM wsystem 8 System name of computer system environment within which this transaction is running. IEF1-MSG wmsg 32 Message area. Used to display messages in message field on GTD panels. Field is further subdivided into MSG ID and MSG test fields. IEF1-MSGID wmsgid 7 Message ID, 6 bytes plus one null character. Used as an identifier of a message. Allows user to read the ID of message being displayed. IEF1-MSGTX wmsgtx 25 Message text area for display on panels. IEF1-COMMAND wcommand 23 Command field. Used for accepting inputs on GTD panels IEF1-SELECT1 wselect1 8 Select field. Used as a work area for GTD Menu Manager for parsing the top level select field on a multi-level menu selection command. IEF1-SBLECT2 wselect2 8 Select field. Used as a work area for GTD Menu Manager for parsing the second level field on a multi-level menu selection command. IEF1-SELECT3 wselect3 8 Select field. Used as a work area for the GTD Menu Manager for parsing the third level select field on a multi-level menu selection command. IEF1-SELECT4 wselect4 8 Select field. Used as a work area for GTD Menu Manager for parsing the third level select field on a multi-level menu selection command. IEF1-CURROW wcurrow 2 16 bit binary number representing row number of cursor position on a video dispiay device. Row 1 is top row. IEF1-CURCOL wcurcol 2 16 bit binary number representing column number of cursor position on a video display device. Column 1 is left-most Column. IEF1-FUNKEY wfunkey 2 16 bit binary number representing function key pressed to enter transaction. Field is set to zero when ENTER key pressed. IEF1-CONFIRM wconfirm 1 Application available field to confirm actions by terminal user. IEF1-SYSTYPE wsystype 1 Host system type field. Valid types listed below: TYPE SYSTEM I IMS C CICS T TSO U UNIX V VMS O OS2 IEF1-CURDT wcurdt 18 Holds date and time input was received in following displayable format: YY/MM/DD HH:MM:MM IEF1-RSVRD1 wrsvrdl 6 Reserved. IEF1-TIME wtime 4 Time input was received. In standard TIME SVC format: X'HHMMSSTH' IEF1-DATE wdate 4 Date input was received. In standard TIME SVC format: XX'OOYYDDDF' IEF1-YEAR wyear 2 Year input was received. In 16 bit binary form, for example 1988 = `88,` 1989 = `89.` IEF1-MON wmon 2 Month input was received. In 16 bit binary form, for example Jan = `1 ,` Feb = `2 .` IEF1-DAY wday 2 Day input was received. In 16 bit binary form. IEF1-HELFBLK whelpblk 2 Reserved. IEF1-TEXA wtext 1600 Array work area, twenty 80 character rows of text information suitable for display in GTD HELP, GLOSSARY, and INFO panels. Used by GTD system functions to present these data items. IEF1-HELPPNL whelppnl 8 Contains name of panel HELP support facility is preparing help information displays for. IEF1-HELPCHP whelpchp 8 Contains TIOLR chapter name where HELP documentation is being derived from. IEF1-HELPSEC whelpsec 8 Contains TIOLR section name where HELP documentation is being derived from. IEF1-HELPPAG whelppag 8 Contains TIOLR page name HELP documentation is being derived from. IEF1-GLOSSEC wglossec 8 Contains TIOLR section name where GLOSSARY documentation is being derived from. This is target word for a GLOSSARY selection. Can be modified by found name within GLOSSARY chapter. IEF1-GLOSPAG sglospag 8 Contains TIOLR page name where GLOSSARY documentation is being derived from. IEF1-GLOSPNL wglospnl 8 Contains name of panel interrupted for GLOSSARY information presentation. IEF1-INFOCHP winfochop 8 Contains TIOLR chapter name presented through INFO. IEF1-INFOSEC winfosec 8 Contains TIOLR section name presented through INFO. IEF1-INFOPAG winopag 8 Contains TIOLR page name presented through INFO. IEF1-PROC wproc 8 Procedure name of active GTD procedure. IEF1-BELL wbell 1 Used in preparation of output displays. Allows application to indicate the bell is to ring on an output display for a 3270 device. Field set to Y indicates bell is to ring. IEF1-OPTIN woption 3 Used for selecting function key prompts for terminals without function keys. IEF1-NAPPL wnappl 4 Reserved. IEF1-NSYSTEM wnsystem 8 New system name. Activated as result of procedure executing with IEF1-XSTATE of LINK. IEF1-NTRAN wntran 8 Name of transaction to be executed as result of IEF1- XSTATE of LINK. IEF1-NPROC wnproc 8 Name of procedure to be executed as result of IEF1-XSTATE for a procedure specifying LINK. IEF1-XSTATE wxstate 8 Exit state for an application. Can be set to LINK, RETURN, RESET, QUIT, indicating action to be taken by system as a result of the procedure completion. IEF1-RSVRD3 wsvrd3 16 Reserved. IEF1-DATA1 wdata1 8 Generic work field available to GTD application. IEF1-DATA2 wdata2 8 Generic work field available to GTD application. IEF1-DATA3 wdata3 8 Generic work field available to GTD application. IEF1-DATA4 wdata4 8 Generic work field available to GTD application. IEF1-OUTPUT woutput 32 Generic work area available to GTD application. IEF1-ETIMINT wetimint 4 Number of microseconds (in integer format) required for transaction initialization. IEF1-ETIMGSY wetimgsy 4 Number of microseconds (in integer format) required to execute `GET SYSTEM` request from IEC. IEF1-ETIMGUY wetimgu 4 Number of microseconds (in integer format) required to execute the `GET UNIQUE` to IMS. IEF1-ETIMGDV wetimgdv 4 Number of microseconds (in integer format) required to execute `GET DEVICE` request from IEC. IEF1-ETIMGVW wetimgvw 4 Number of microseconds (in integer format) required to execute `GET VIEW` request from IEC. IEF1-ETIMRVW wetimrvw 4 Number of microseconds (in integer format) required to execute `RESTORE VIEW` request from IEC. IET1-ETIMITM wetimitm 4 Number of microseconds (in integer format) required to complete input TMS call. IEF1-ETIMXC wetimxc 4 Number of microseconds (in integer format) required to execute procedure. IEF1-ETIMTOT wetimtot 4 Number of microseconds (in integer format) required to execute total transaction from transaction initialization through procedure execution. IEF1-ETIM36 weit36 4 Reserved time field. IEF1-ETIM40 weit40 4 Reserved time field. IEF1-ETIM44 weit44 4 Reserved time field. IEFI- wlookahead 64 Used by application to allow LOOKAHEAD user to select new transaction. If first byte is indicated, the menu request is ignored by IET. Application has responsibility to take action to select next (LOOKAHEAD) action according to contents of this field. IEF1-RSVRDS wrsvrds 400 Reserved.

L7: Entry 2 of 3 File: USPT Aug 16, 1994

DOCUMENT-IDENTIFIER: US 5339392 A
TITLE: Apparatus and method for creation of a user definable video displayed document showing changes in real time data

BSPR:
An apparatus and method according to the teachings of the invention provides a computer facility (hereafter the application or program) whereby a user, using a collection of layout tools may define an active document. "Active document" as that term is used herein means a video displayed document of one or more "sheets" of the user's design which incorporates text, displays of real time data in user definable style, e.g., color, font, background, pen size etc. and format, e.g., quote, ticker, graph etc., alarms, and alarm scripts, i.e., user defined scripts of commands to be processed (much like a word processing or spreadsheet macro)when an alarm limit is exceeded. The program automatically accesses the network to which the host is connected through network interface software which establishes the proper subscriptions for the desired real time data with the appropriate information service and the server upon which this service process is running. Real time data is then passed to the program from whatever network communication process is being used and is immediately displayed in the format, style and location previously specified by the user. Although the invention will hereafter be described in the preferred embodiment for use in a financial environment such as a trading floor of a broker such as Dean Witter etc., the invention is not limited to such applications. Any complex system which generates real time data which control operators must monitor are subject to being monitored and controlled using the teachings of the invention.

BSPR:
In the preferred embodiment, the tools available for defining an active document are as follows. A label tool allows the user to enter static text to label or annotate the active document or to create his or her own personalized help screens. A quote tools displays the value of an issue, including a user defined set of other fields pertaining to that particular company in a display style specified by the user. For example, a brief style displays only the price where a comprehensive style displays all the available fields. A ticker tool can be used as a selective or block ticker, and can show data in any display style. Upticks and Downticks can be shown in color and volume information can be included. A page fragment tool displays a region of a page-based feed such as Telerate or Reuters. Any region of the page designated by the user can be displayed from a single character to a full page. Highlighting modes are provided to highlight that has changed. A time based graph tool can be used to create graph display objects to graphically display the changes in value of a variable such as price per time. The time and price axes may be scaled to minutes or seconds, and the price value may be set to any unit such as 1/8 or 1/32 of a dollar. Above and below channel segments can be drawn on the graph. Graphs may be merged to show two issues against the same time axis. A data set graph tool can be used to create graph display objects which display the values of multiple instruments such as stocks or bonds or other subscribed values in real me such as a yield curve in a semiconductor processing application environment. Graphs may be merged to show two different sets of issues against each other to indicate market opportunities. A table tool can be used to create display objects which show position blotters, currency lookup tables, and names of commonly used pages securities. A publisher tool publishes information constructed using the invention or entered the user onto the network using the network communication process running in the environment in the invention is running. The published information can be used by other processes linked to the network or as a bulletin board for use by her traders. A button tool can be used to create splay objects that execute scripted actions when the button is "pushed", i.e., selected in any way on the splay such as by clicking on the button by a mouse. The scripted actions are entered by the user in whatever sequence is desired in a language such as the MarketScript.TM. command language comprised of all commands at the invention can execute. In the preferred embodiment, the scripted command sequence can also include commands to the operating system, the network communication software and other processes running on the same host or elsewhere on network. Buttons can be programmed to carry out commonly performed operations such as moving quickly to an important page or performing an operation to be carried out when an alert condition

occurs. The buttons allow creation of hypertext links between different sheets, and the alert scripts can perform operations such as changing a color, flashing an object, sounding an audible alarm or executing an external program. The latter capability provides great flexibility by enabling clients to program features such as the ability to telephone a beeper service when an alert occurs. A "glossary" facility allows users to add new operations, i.e., commands, to the scripting language and customize the menus as needed.

DEPR:
The program according to the teachings of the invention use objected oriented programming style. Although the preferred embodiment of a program according to the teachings of the invention has been written in C language for easier portability among machines using programming conventions to make the C language act like an object oriented programming language, it is easier to construct the program using object oriented programming languages such as C++.

DEPR:
"Active document" as that term is used herein means a video displayed document of one or more "sheets" of the user's design which incorporates text, displays of real time data in user definable style, e.g., color, font, background, pen size etc. and format, e.g., quote, ticker, graph etc., alarms, and alarm scripts, i.e., user defined scripts of commands to be processed (much like a word processing or spreadsheet macro) when an alarm limit is exceeded. The program automatically accesses the network to which the host is connected through network interface software which establishes the proper subscriptions for the desired real time data with the appropriate information service and the sever upon which this service process is running. Real time data is then passed to the program from whatever network communication process is being used and is immediately displayed in the format, style and location previously specified by the user. Although the invention will hereafter be described in the preferred embodiment for use in a financial environment such as a trading floor of a broker such as Dean Witter etc., the invention is not limited to such applications. Any complex system which generates real time data which control operators must monitor are subject to being monitored and controlled using the teachings of the invention.

DEPR:
A label Active Object is just a fixed character string placed in a position on a sheet entered by the user. Labels are used to identify sheets, regions on sheets, and individual monitoring Active Objects as well as in script files to generate messages when an alarm event occurs or to generate customized help screen for a particular active document. A label does not change in real time. Its attributes are: String (field) which is the text string to be displayed in the label object; and, Alignment (radio button list) which is one of three formatting options--left, center and right.

DEPR:
Quotes have four different scripts which are run for different reasons with respect to alerts. FIG. 2 illustrates states behind this concept. A quote is either in the normal state 27 or the alert state 28. When a real time data update comes into a normal state quote and does not trigger an alert, the "normal update" script is run. A script is a user defined string of commands that are executed in sequence. They can be commands that the program of the invention understands, commands to the operating system or other processes operating in the environment or commands to any other process running anywhere else on the network. The script language also has a glossary facility facility whereby the user can define new commands and add them to the script language. When an update comes in which triggers an alert, the "begin alert" script 30 is run. This script takes the quote object into the alert state 28 where the "alert update" script is run. Additional updates which are in alert run will continue to run the "alert update" script. Then when an update comes in which is back in the normal range, the "end alert" script will be run, followed by the "normal update script. Thus, the four scripts provide a way of checking for changes in the state, or for staying in the same state.

DEPR:
A button is an object which the user can interact with, and will cause a script to be carried out when clicked on. For instance, the button might perform the equivalent of a Sheet Next command, or transfer the user to a specific sheet. Buttons allow the user to determine the dynamic action of the sheets, as well as their appearance. These scripts are expressed in the MarketScript.TM. language.

DEPR:
There are roughly 80 pixels per inch on a workstation screen. All of the x, y, width, and height measures in the scripting language are in pixels.

DEPR:
The Event Trigger is a specification of conditions under which the user wishes to do extra processing on the Active Object. For example, the user can set alarm limits such

as a certain price or trading volume for a particular quoted Active Object, and when a real time data update indicates that the limit has been exceeded, an alarm condition exists to transfer the Active Object from the normal update state to the alert state. The Event Script of commands to execute upon occurrence of the specified alarm condition is specified in the Event Script specifications shown generally at 110. The things that can be scripted to happen upon occurrence of an alarm condition are limited only by the imagination of the user. Minimally, the script may specify an audible beep and/or a change in color of an Active Object. More exotic scripts may issue commands on the network to start another process running to dial a beeper, issue a sell order, issue a buy order, etc. Other scripts may publish some or all the data on one or more sheets of an active document on the network, etc. The commands in the scripting language generally include all the commands understood by the script processor as well as commands defined by the user and can, in some embodiments, include commands to the operating system, the high level network interface or other processes running on the network. Generally the commands understood by the script processor will include the name of the object, the desired operation and an argument, i.e., what value to set etc.

DEPR:
After the environment is initialized, the global dispatcher 136 has control of the system. It then waits for an event and processes each event appropriately. For example, a mouse event in the form of a click on a frame object icon such as a window meaning move this window to the top of the stack will be dispatched to the frame object 52 by calling the appropriate operation to move this window to the top of the stack. A mouse click on a menu bar option as shown at 12 in FIG. 1 would be dispatched to the menu manager 54 as a call to the operation to display the appropriate pop-up menu for the suboptions of the selected menu option.

DEPR:
Button objects provide great flexibility. A button object can be scripted to pop up on a user written help screen, make another Active Object appear or disappear, or do any other commands or sequence of commands within the scripting language.

DEPR:
The user can define a sequence of actions to be carried out when a button is pressed or a price update occurs. These actions are expressed in a macro language called MarketScript.about.. This facility greatly increase the flexibility and generality of the program.

DEPR:
Dynamic graphs chart securities prices and other TIB.RTM. subjects in real-time. Multiple subjects can be included in a graph and simple arithmetic operations can be performed on the axes, such as spreads. Graphs can also be merged, meaning that two prices can be shown against one time axis.

DEPR:
Buttons are triggers for scripts which are carried out when clicked. These scripts are expressed in the MarketScript.TM. language. In most cases, these actions are similar to those inaccessible through the menus, but these triggers can be placed on the sheets themselves. For instance, the user can create a button which will bring up a particular sheet when clicked on. This allows the creation of "hypertext links" between related information, such as a security and its options pricing. Available tools include:

DEPR:
When there are more than one selected object, one is marked as the Traversal object, meaning that keyboard input will be sent to it. This facility is mostly used for quick editing of the symbols in Quote objects, or the current page shown in a Page Fragment, which are explained in more detail below.

DEPR:
In order to refer to specific objects or collections of objects in the MarketScript.RTM. language used by the scripts, objects can be given names. Note that the name of a label, for instance, is different from the text shown by the label. Choose

DEPR:
The data-exchange component of the communication interface according to the teachings of the TIB.RTM. software includes a forms-manager module and a forms-class manager module. The forms-manager module handles the creation, storage, recall and destruction of instances of forms and calls to the various functions of the forms-class manager. The latter handles the creation, storage, recall, interpretation, and destruction of forms-class descriptors which are data records which record the format and semantic information that pertain to particular classes of forms. The forms-class manager can also receive requests from the application or another component of the communication interface to get a particular field of an instance of a form when identified by the name or meaning of the field, retrieve the appropriate form instance, and extract and deliver

the requested data in the appropriate field. The forms-class manager can also locate the class definition of an unknown class of forms by looking in a known repository of such class definitions or by requesting the class definition from the forms-class manager linked to the foreign process which created the new class of form. Semantic data, such as field names, is decoupled from data representation and organization in the sense that semantic information contains no information regarding data representation or organization.

DEPR:
Returning to the consideration of the Player.sub.-- Name class definition of FIG. 22, the second field is named Age. This field contains forms of the primitive class named Integer associated with class number 12 and defined in FIG. 25. The Integer class of form, class 12, has, per the class definition of FIG. 25, a data representation specification of Integer.sub.-- 3, meaning the field contains integer data having three digits. The last two fields of the class 1000 definition in FIG. 22 are Last.sub.-- Name and First.sub.-- Name. Both of these fields contain primitive forms of a class named String.sub.-- Twenty.sub.-- ASCII, class 10. The class 10 class definition is given in FIG. 25 and specifies that instances of forms of this class contain ASCII character strings which are 20 characters long.

DEPR:
The class definition of FIG. 24 shows how nesting of forms can occur in that each field of a form is a form itself and every form may be either primitive and have only one field or constructed and have several fields. In other words, instances of a form may have as many fields as necessary, and each field may have as many subfields as necessary. Further, each subfield may have as many sub-subfields as necessary. This nesting goes on for any arbitrary number of levels. This data structure allows data of arbitrary complexity to be easily represented and manipulated.

DEPR:
The next step is symbolized by block 300. This step involves accessing the form identified in the original format conversion call and searching through the form to find the first field containing a primitive class of form. In other words, the record is searched until a field is found storing actual data as opposed to another constructed form having subfields.

DEPR:
Block 306 in FIG. 28 symbolizes the process of calling the conversion program identified by step 304 and performing this conversion routine to change the contents of the field selected in step 300 to the "to" or target format identified in step 302. In the hypothetical example, the routine ASCII.sub.-- ETHER would be called and performed by step 306. The call to this routine would deliver the actual data stored in the field selected in the process of step 300, i.e., field 242 of the instance of a form shown in FIG. 26, such that the text string "U.S. Open" would be converted to a packed ETHERNET.TM. format.

DEPR:
After locating the field of interest in the class definition, the class manager returns a relative address pointer to the field of interest in instances of forms of this class. This process is symbolized by block 326 in FIG. 32. The relative address pointer returned by the class manager is best understood by reference to FIGS. 22, 24 and 26. Suppose that the application which made the Get.sub.-- Field call was interested in determining the age of a particular player. The Get.sub.-- Field request would identify the address for the instance of the form of class 1002 for player Blackett as illustrated in FIG. 26. Also included in the Get.sub.-- Field request would be the name of the field of interest, i.e., "age". The class manager would then access the instance of the form of interest and read the class number identifying the particular class descriptor or class definition which applied to this class of forms. The class manager would then access the class descriptor for class 1002 and find a class definition as shown in FIG. 24. The class manager would then access the class definitions for each of the fields of class definition 1002 and would compare the field name in the original Get.sub.-- Field request to the field names in the various class definitions which make up the class definition for class 1002. In other words, the class manager would compare the names of the fields in the class definitions for classes 10, 1000, and 1001 to the field name of interest, "Age". A match would be found in the class definition for class 1000 as seen from FIG. 22. For the particular record format shown in FIG. 26, the "Age" field would be the block of data 262, which is the tenth block of data in from the start of the record. The class manager would then return a relative address pointer pointing to the "age" field as symbolized by the processing in block 326 of FIG. 32. This relative address pointer is returned to the client application which made the original Get.sub.-- Field call. The client application then issues a Get.sub.-- Data call to the forms-manager module and delivers to the forms-manager module the relative address of the desired field in the particular instance of the form of interest. The forms-manager module must also know the address of the instance of the form of interest which it will

already have if the original Get.sub.-- Field call came through the forms-manager module and was transferred to the forms-class manager. If the forms-manager module does not have the address of the particular instance of the form of interest, then the forms manager will request it from the client application. After receiving the Get.sub.-- Data call and obtaining the relative address and the address of the instance of the form of interest, the forms manager will access this instance of the form and access the requested data and return it to the client application. This process of receiving the Get.sub.-- Data call and returning the appropriate data is symbolized by block 128 in FIG. 32.

DEPR:
Normally, class-manager modules store the class definitions needed to do semantic-dependent operations in RAM of the host machine as class descriptors. Class definitions are the specification of the semantic and formation information that define a class. Class descriptors are memory objects which embody the class definition. Class descriptors are stored in at least two ways. In random access memory (RAM), class descriptors are stored as forms in the format native to the machine and client application that created the class definition. Class descriptors stored on disk or tape are stored as ASCII strings of text.

DEPR:
(2) Presentation: Presents structured data in proper form for use by application programs. Provides a set of services which may be selected by the application layer to enable it to interpret the meaning of data exchanged.

DEPR:
Our notion of a form is more fundamental, akin to such basic notions as record or array. Our notion takes its meaning from the original meaning of the Latin root word forma. Borrowing from Webster: "The shape and structure of something as distinguished from its material". Forms can be instantiated, operated on, passed as arguments, sent on a network, stored in files and databases. Their contents can also be displayed in many different formats. "templates" can be used to specify how a form is to be displayed. A single form (more precisely, a form class) can have many "templates" since it may need to be displayed in many different ways. Different kinds of users may, for example, desire different formats for displaying a form.

DEPR:
Each form is a member of a specific form class. All forms within a class have the same fields and field's labels (in fact, all defining attributes are identical among the forms of a specific class). Each form class is named and two classes are considered to be distinct if they have distinct names (even though the classes may have identical definitions). Although the forms software does not assign any special meaning or processing support to particular form names, the applications using it might. (In fact, it is expected that certain form naming conventions will be established.)

DEPR:
Clients that are interested in pointer semantics should use forms of the basic type Form Pointer and the function Form.sub.-- set.sub.-- data.sub.-- pointer. Forms of type Form Pointer contain only a pointer to a form; hence, pointer semantics is used for assignment. Note that the C programming language supports pointer semantics for array assignment.

DEPR:
A sub-form or field of a constructed form can be accessed by its field name or by its field identifier (the latter is generated by the Forms package). The name of a sub-form that is not a direct descendent of a given form is the path name of all the fields that contain the requested sub-form, separated by dot. Note that this is similar to the naming convention of the C language records.

DEPR:
5.6. Form-class Definition Language

DEPR:
Form classes are specified using the "form-class definition language," which is illustrated below. Even complex forms can be described within the simple language features depicted below. However, the big attraction of a formal language is that it provides an extensible framework: by adding new language constructs the descriptive power of the language can be greatly enhanced without rendering previous descriptions incompaTIB.RTM.Ie.

DEPR:
There follows a list of definitions of some of the words and phrases used to describe the TIB.RTM. software.

DEPR:
Class/Class Descriptor/Class Definition: A definition of the structure and organization of a particular group of data records or "forms" all of which have the same internal representation, the same organization and the same semantic information. A class descriptor is a data record or "object" in memory that stores the data which defines all these parameters of the class definition. The Class is the name of the group of forms and the Class Definition is the information about the group's common characteristics. Classes can be either primitive or constructed. A primitive class contains a class name that uniquely identifies the class (this name has associated with it a class number or class.sub.-- id) and a specification of the representation of a single data value. The specification of the representation uses well known primitives that the host computer and client applications understand such as string.sub.-- 20 ASCII, floating point, integer, string.sub.-- 20 EBCDIC etc. A constructed class definition includes a unique name and defines by name and content multiple fields that are found in this kind of form. The class definition specifies the organization and semantics or the form by specifying field names. The field names give meaning to the fields. Each field is specified by giving a field name and the form class of its data since each field is itself a form. A field can be a list of forms of the same class instead of a single form. A constructed class definition contains no actual data although a class descriptor does in the form of data that defines the organization and semantics of this kind of form. All actual data that define instances of forms is stored in forms of primitive classes and the type of data stored in primitive classes is specified in the class definition of the primitive class. For example, the primitive class named "Age" has one field of type integer.sub.-- 3 which is defined in the class definition for the age class of forms. Instances of forms of this class contain 3 digit integer values.

DEPR:
Field: One component in an instance of a form which may have one or more components each named differently and each meaning a different thing. Fields are "primitive" if they contain actual data and are "constructed" if they contain other forms, i.e., groupings of other fields. A data record or form which has at least one field which contains another form is said to be "nested". The second form recorded in the constructed field of a first form has its own fields which may also be primitive or constructed. Thus, infinitely complex layers of nesting may occur.

DEPR:
Semantic Information: With respect to forms, the names and meanings of the various fields in a form.

DEPR:
Service: A meaningful set of functions or data usually in the form of a process running on a server which can be exported for use by client applications. In other words, a service is a general class of applications which do a particular thing, e.g., applications supplying Dow Jones News information. Quotron data feed or a trade ticket router. An application will typically export only one service, although it can export many different services.

DEPR:
The basic process described in the flow charts to be described below and embodied in the source code appendix attached hereto is generally to use style maps associated with every display object and the metadata stored within every self-describing data object to control a selection, formatting and display process. Each self-describing data object is an aggregate comprised of one or more items of fixed and/or real time data (data which can change over time) and metadata which describes one or more of the following things about the data of the self-describing data object: the organization or format of the data in the fields of the self-describing data objects; the representation or type of the data in the data fields; and/or the element or field names for the data elements or fields of the self-describing data object. The style maps are essentially filter specifications which define which of the multiple fields of a multi-field, self-describing data object to select for display. The process controlled by the combination of style map and metadata is a process of selecting only user designated items to be displayed per the style map associated with the display object defining where and how data from a self-describing data object is to be displayed and for automatically converting the selected data items from certain fields of a self-describing data object to a proper format for display using the metadata of the self-describing data object aggregate as a pointer to the proper conversion/formatting software routine and for displaying the converted data. This process happens every time there is a change in the style map name, and every time a new self-describing data object arrives carrying an update to the value of some item of data displayed on the "living document" for which there is an open subscription.

DEPR:
Step 962 represents the process of converting the value received by the extraction process of step 960 into a set of characters for storage in a working buffer This done

according to the conversion routine indicated by the metadata and the formatting specifications indicated by the style map element being processed. For example, the metadata might indicate that the extracted value is a floating point number, and the style map element might indicate that for the number to be displayed in the display object being processed requires two decimal places of precision. Thus, the conversion routine chosen will be that which is appropriate for floating point to character conversion, and the number of decimal places, i.e., two, will be given as a parameter to this routine. Other examples would how to handle extracted values which are dates or integers.

DEPC:
SCRIPTING <u>LANGUAGE</u>